計算機アーキテクチャ 70－4
（１９８８．３．１０）

並列処理システム－晴－の要素プロセッサ構成

山名早人 丸島敏一 草野義博 村岡洋一

早稲田大学 理工学部

－晴－は、データフロー実行とコントロールフロー制御を融合した科学技術計算用のアレイプロセッサである。本稿では、－晴－の構成要素である要素プロセッサの構成及び動作について述べる。要素プロセッサ内では、処理の高速化のために（１）定型的な配列演算に対するベクトル処理、（２）データフロー実行において相手データを待ち合わせるための待ち合わせ記憶のバンク化、（３）待ち合わせ記憶の裏バンク化を行っており、ソフトウエアシミュレーションによってこれらの機能の有効性を確認した。

A Construction of Processing Element in a Parallel Processing System -Harray-

(in Japaneses)

Hayato YAMANA, Toshikazu MARUSHIMA,
Yoshihiro KUSANO and Yoichi MURAOKA

School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjuku, Tokyo, 160 JAPAN

A parallel processing system -Harray- is an array processor aiming at fast execution of scientific calculation which has both data flow mechanism and control flow mechanism.

In this paper, a construction of its processing element and its execution mechanism are described. The processing element has (1) vector processing mechanism for conventional matrix calculation, (2) a banked matching memory unit where is the place to find a partner datum in the data flow execution, and (3) a preliminary matching memory unit for prefetching data. The effectiveness of these three mechanisms is confirmed by software simulation.

１．はじめに

我々は、科学技術計算を高速に処理するために並列処理システム－晴－を提案している[1]。－晴－では、プログラムをマクロブロックというブロックに分割し、マクロブロック同士の上位レベルの並列処理と、マクロブロック内のデータフロー実行という２段階の並列処理を実現している。これにより、データフロー実行での自然な並列性の抽出及びマクロブロック間のコントロールフロー制御による並列性の制御を可能としている。

本稿では、－晴－の構成要素でありデータフロー実行を行う要素プロセッサについて、その構成及び動作を述べる。まず２章で－晴－の全体構成を概説し、３章で要素プロセッサ内の各ユニットについて述べると共に、２つの実行モード（データフローモードとベクトルモード）について比較を行う。そして、４章ではデータフロー実行において、その処理速度が問題となる待ち合わせ記憶の高速化について２つの提案を示し、その評価を行う。

２．－晴－の全体構成

－晴－は図１に示すように６つの大きなユニットからなり、プログラムのコンパイルや初期データのロード及び結果出力のため、ホストコンピュータと接続されている。

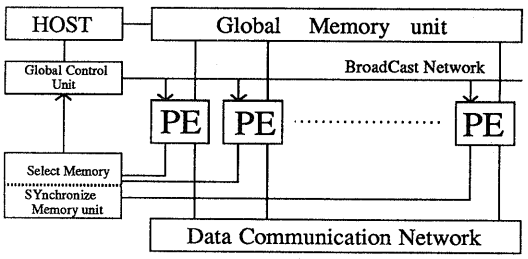


図１ -晴-の全体構成

集中制御機構(GCU:Global Control Unit)は、－晴－システム全体を制御する機構であり、各要素プロセッサ(PE:Processing Element)に対し、マクロブロック実行制御命令を出す。この命令は、起動放送ネットワーク(BCN:BroadCast Network)を通じて各ＰＥに放送される。ＰＥでは、ＧＣＵからの命令に従ってマクロブロックの実行を開始し、その終了を同期メモリ(SYM:SYnchronize Memory)に報告する。このＳＹＭは、複数のＰＥ間同期を高速に実行するためのものであり、ＧＣＵはＳＹＭの同期完了報告をチェックしながら制御を続ける。

ＰＥ間のデータ交換のためには通信ネットワーク(DCN:Data Communication Network)を用意している。また、配列等の大域的なデータは共有データメモリ(GM:Global Memory)に格納し処理を行う。

ＧＣＵによるマクロブロック実行開始（マクロブロックの活性化）方式には、(1)同期実行、(2)先行実行、(3)仮実行の３種類がある[1]。(1)はif文などの条件分岐によりマクロブロックの実行が確定し、必要データが全て揃っていることを保証して実行する方式であり、(2)は(1)の方式に比べ全データの到着を待たずに実行を開始する方式である。また(3)は条件分岐による実行の決定が行われない段階で実行を開始する方式である。

３．要素プロセッサ(PE)の構成とその動作

３．１．要素プロセッサ概要

ＰＥの構成を図２に示す。

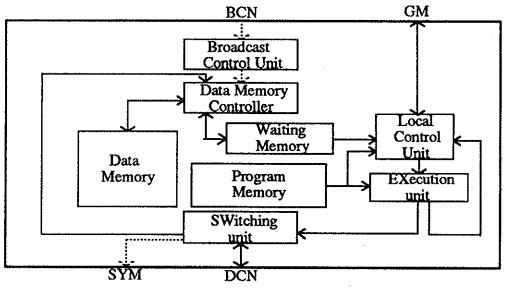


図2 PEの構成

ＰＥ内での実行方式には、(1)データフロー実行を行うデータフローモード、(2)定型的な配列演算を行うベクトルモードがある。

データフローモードでは、ＧＣＵからのマクロブロック実行命令を起動放送制御機構(BCU:BroadCast Control Unit)が受信することにより実行を開始する。まず、データメモリ(DM:Data Memory)内にある起動すべきマクロブロックに属するデータ（以後パケットと呼ぶ）を待ち合わせ記憶(WM:Waiting Memory)に転送し、実行が始まる。WMで対となったパケットは、実行機構(EX:EXecution unit)で演算処理される。その後ＥＸで新しいパケットが作られ、スイッチ機構(SW:SWitching unit)、データメモリ制御機構(DMC:Data Memory Controller)を通って再びWMへ入り処理が続く。なおWMには、現在実行中のマクロブロックに属するパケットのみを入れ、それ以外はＤＭへ格納する。これにより、WM内のパケット数を少なくし、WMの負荷を軽くすることが出来る。

またベクトルモードは、データフローモードにおいてベクトルモード移行命令を実行することにより実行が開始される。このベクトル実行では、ＰＥ内制御機構(LCU:Local Control Unit)がプログラムメモリ(PM:Program Memory)内のプログラムを読み出しながら、ＰＥ全体を制御し実行を行う。この時、ＥＸの出力はWMに送らずにＬＣＵへ直接フィードバックさせ実行する。

３．２．構成ユニット

ＰＥは、図２に示すように８つのユニットからなる。以下、各ユニットについて説明をする。

(a) 起動放送制御機構(BCU:Broadcast Control Unit)

集中制御機構(GCU)から起動放送ネットワーク(BCN)を通じて放送される命令を解読し、自ＰＥ宛のものであればＰＥ内部の各ユニットに対して制御命令を出すための機構である。

(b) 待ち合わせ記憶(WM:Waiting Memory)

データフロー実行時に、パケットの待ち合わせを行うユニットである。－晴－では、マクロブロック単位ごとにデータフロー実行を行っている。このため、プログラム全体に対してデータフロー実行を行う場合に比較してWMに溜まるパケット数を少なく出来、WMの小容量化が可能である[2]。これにより、現在のデバイス技術を用いてフルアソシアティブに本ユニットを構成し、WMを高速化することが可能である。

WMの詳細については、第４章で述べる。

(c) データメモリ(DM:Data Memory)

活性化されていないマクロブロックに属するパケット（不活性パケット）を格納するためのメモリである。このＤＭの管理は全てデータメモリ制御機構(DMC)が行う。また、仮実行時の結果パケットもＤＭに格納する。仮実行されたマクロブロックに制御が実際には移らなかった場合、間違った結果が他のマクロブロックで使用されるのを防ぐため、仮実行時には、他のマクロブロックに対して結果パケットを送ることが出来ない。したがって、一時的にＤＭに格納し、実際に実行許可がおりた段階で外部に送り出す。

(d) データメモリ制御機構(DMC:Data Memory Controller)

ＤＭＣの構成図を図３に示す。ＤＭＣは３つの大きな機能を持っている。これらは、(1)パケットが現在活性中のマクロブロックに属しているか否かを判断し、活性中のものをＷＭに、そうでないものをＤＭに振り分ける機能（パケット選別機構）、(2)マクロブロック起動時に、起動マクロブロックに属するパケットをＷＭに転送する機能（パケット転送機構）、(3)ＤＭ内に格納されるパケットを管理する機能（ＤＭ管理機構）である。以下、それぞれの機能について説明する。

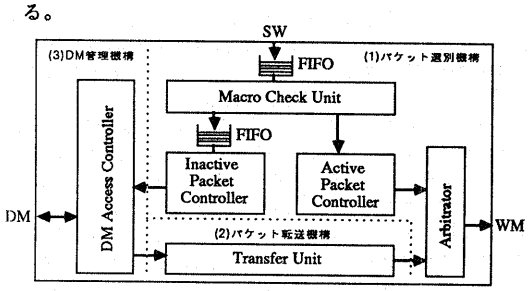


図3 DMCの構成

(1) パケット選別機構

到着パケットの所属マクロブロック番号と、現在実行中のマクロブロック番号（活性マクロブロック番号）を比較し、一致すればＷＭに送出し、一致しなければＤＭに格納する。前者は現在実行中のマクロブロックに属するパケット（活性パケット）に関する制御であるため、高速性が要求される。一方後者の場合は、不活性パケットであるため、ＤＭへの格納処理に高速性は要求されない。このため、図３で示すように活性パケット処理部と不活性パケット処理部を独立させ、不活性パケットの処理によって活性パケットの処理が遅くならないようにしている。

(2) パケット転送機構

不活性パケットはＤＭに格納されている。このため、マクロブロック起動時に必要となるパケット、すなわち起動マクロブロックに属するパケットをＤＭからＷＭへ転送する必要がある。これを行うのがパケット転送機構である。

(3) ＤＭ管理機構

マクロブロック起動時には、マクロブロック単位でパケットをＷＭに高速転送しなければならない。このためＤＭにおけるパケット管理方式も、これに適したものである必要がある。そこで、次のようなパケットの管理方式をとる。まず、到着したパケットの属するマクロブロック番号を調べ、そのマクロブロックの領域が既にＤＭ上に確保されているかどうかを調べる。まだ確保されていなければ、コンパイラによって計算されているそのマクロブロックの大きさの領域をＤＭ上に確保し、その領域内にパケットを格納する。このように、パケットをマクロブロックごとに管理する方式では、領域がＤＭ上に確保されていない場合に処理時間がかかる。しかし、ＤＭに格納されるパケットは不活性なパケットであり、格納に時間がかかっても問題はない。これに対してＷＭ転送時には、パケットがマクロブロックごとに格納されているため、パケットのサーチが必要なく高速に転送を行うことが出来る。

(e) プログラムメモリ(PM:Program Memory)

データフローモードでは、各ノード実行後の行き先や命令などの次のノードに関する情報を格納し、ベクトルモードでは、ベクトル実行時の命令を格納する。

(f) 実行機構(EX:EXecution unit)

ＥＸは演算を実行すると共に、データフローモードでは、次のノードに関する情報をＰＭから読み出し新しいパケットを作る。ＥＸの構成図を図４に示す。

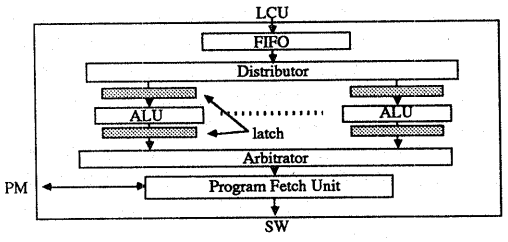


図4 EXの構成

図４に示すように、ＥＸは複数のＡＬＵからなる。これは、ＥＸに到着したパケットを効率よく処理するためである。複数のＡＬＵを用いた場合、パケット（負荷）の分散方法が問題となるが、－晴－では図４に示すように各ＡＬＵの前後に１段のラッチを設け、各ＡＬＵ及びラッチの内部状態によって最も負荷の小さいＡＬＵへ割り当て、効率のよい分散を実現している[3]。

データフロー実行時には、命令実行後、次のノードに関する情報をＰＭからフェッチし新しいパケットを作る必要がある。－晴－では、Program Fetch Unit(以後フェッチユニットと言う）がこれを行う。また、演算の結果が複数のノードで使用される場合には、データ部分が同じで行き先情報の異なったパケットが、複数個必要となる。これに対し、－晴－では次の２つの方法を用意している。

(1) コピー命令（コピーノード）を新たに設け実行する方法
(2) コピー命令を用いずに複数個のパケットをフェッチユニットで作成する方法

(1)の方法を用いた場合は、コピー命令実行のために、ＥＸ→ＷＭ→ＥＸといったループをパケットが通らなければならず、このループ通過に伴うオーバーヘッドが生じる。これに対して(2)の方法では、(1)のオーバーヘッドは生じない。しかし、多数のコピーを行う場合にはフェッチユニットでの処理時間が大きくなり、フェッチユニットの前で他のパケットが待たされ、遅延が生じる。特にＡＬＵに高負荷がかかっている時、このオーバーヘッドは大きくなり、データフローグラフのクリティカルパス上にある演算の実行を遅らせてしま

う可能性がある。しかし、ＡＬＵの負荷が小さい時には、フェッチユニット前で待たされるパケット数は少なく、(1) の方法よりも(2)の方法の方が有利となる。したがって－晴－では、これら２つの方法をうまく組み合わせて使用することを検討中である。また、高負荷時においても、コピー数が平均３以下であれば、フェッチユニットでの処理時間増加に伴うオーバーヘッドは、小さいことがシミュレーションにより確認されている(3)。

(g) ＰＥ内制御機構(LCU:Local Control Unit)

構成図を図５に示す。

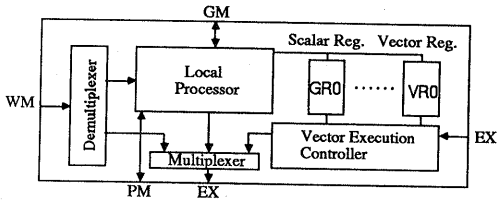


図5 LCUの構成

ＬＣＵは、ＧＭとのインタフェースの役割を果たしている。WMからのパケットは、Demultiplexerにより、ＬＣＵ内で処理されるもの（ＧＭアクセス関係の命令）と、そうでないものとに選別される。前者は、Local Processorへ送られ処理されるが、後者はそのまま Multiplexerを通ってＥＸへ送られる。このため、ＧＭアクセス関係の命令とその他の命令は、ＬＣＵとＥＸで並列に処理出来る。

ベクトルモード時においては、ＬＣＵはベクトル処理全体を制御する制御機構として動作する。また、図５に示すようにベクトルレジスタを設けており、ベクトル処理の高速化をはかることが出来る。

(h) スイッチ機構(SW:Switching unit)

ＳＷは、ＰＥとＤＣＮ間のインタフェースであり、他ＰＥへＤＣＮを介してパケットを送出したり、他ＰＥから自ＰＥ宛のパケットを受け取ったりする役目を果たす。また、マクロブロック終了をＳＹＭに対して報告するのも本ユニットの機能である。

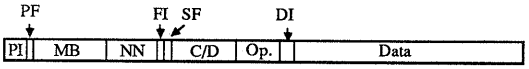


図6 基本パケット形式

表1 各フィールドの内容

フィールド名	ビット数	説明
PI(Packet Id.)	3	パケットの種類及び実行ユニットを示す 0:初期ロードパケット 1:仮実行の出力パケット 4:EXで実行　6:SWで実行 5:LCUで実行　7:WMで実行
PF(Pre-exec. Flag)	1	借実行結果パケット識別フラグ
MB(Macro Block No.)	10	所属マクロブロック番号
NN(Node No.)	7	行き先ノード番号
FI(Field Id.)	1	C/Dフィールドの内容を示す 0:C/Dフィールドはカラー 1:C/Dフィールドは行き先PE番号
SF(Sticky Flag)	1	スティッキートークン識別フラグ
C/D(Color/Distination PE No.)	9	カラー/行き先PE番号
Op.(Operation)	6	実行命令
DI(Data Input Id.)	2	データ識別子 0:単一実行データ　1:右入力 2:左入力
Data	32	データ

３．２．パケット形式と命令セット

ここでは、－晴－で用いるパケットの形式及び命令セットについて説明する。

３．２．１．パケット形式

図６に基本パケットの形式を示す。１パケットを７２bitで構成し、各フィールドの内容は表１に示す通りである。

３．２．２．命令セット

表２にＰＥにおける命令セットを示す。命令セットは、データフローモード用とベクトルモード用の２種類に大きく分類される。

データフローモード用の命令は、(1)算術論理演算命令、(2)プレディケイト命令、(3)フロー制御命令、(4)共有メモリアクセス命令、(5)その他、の５種類に分類される。－晴－では、データフローモード／ベクトルモードの切り換えのためにベクトルモード移行命令(VON)が、マクロブロックの終了報告を行うためにマクロブロック終了命令(FIN)が、通常の命令以外に付け加えられている。

ベクトルモードでは、ＬＣＵがＰＥ全体を制御して実行を行う。このベクトルモードでは、その演算実行においてデータフローモードでの演算結果も必要となるため、モード間でのデータの受け渡しが必要である。しかし、データフロー実行では、パケット自信がデータを持ちＰＥ内を動き回ることによって演算が実行され、レジスタという概念がないので、データの受け渡し方法が問題となる。このため、－晴－ではレジスタストア命令(RSTR)によってＬＣＵ内に設けられたレジスタにデータをストアし、データフローモードからベクトルモードへのデータの受け渡しを行っている。

ベクトルモード用の命令は、(1)算術論理演算命令、(2)プレディケイト命令、(3)共有メモリアクセス命令、(4)レジスタセット命令、(5)順回演算処理命令、(6)その他、に分類される。ベクトルモードは、ループ演算のようにあらかじめデータの依存関係の解析が行え、かつＧＭにそのデータがおかれている場合の処理高速化を目的としている。特に、(5)の順回演算処理は、A(I)=A(I-1)+B(I) のように直前の演算結果を用いて処理が進んでいくループに対する命令である。この例については、第3.3.3節で詳しく述べる。また、データフローモードからベクトルモードへの移行命令があるように、ベクトルモードからデータフローモードへ再び戻るための命令(VFIN)が用意されている。

表2 -晴-の命令セット

Data Flow Mode

(1)算術論理演算
ADD SUB MUL DIV FADD FSUB FMUL FDIV SFIL SFIR
INT REAL INC DEC AND OR NOT XOR

(2)プレディケイト命令
GE GT LE LT EQ NEQ FGE FGT FLE FEQ FNEQ

(3)フロー制御命令
T F SYNL SYNR SYNC CINC CDEC STC KSTC

(4)共有メモリアクセス命令
MRGP DSTR DLD

(5)その他
PSET RSTR CP VON FIN

Vector Mode

(1)算術論理演算
VADD VSUB VMUL VDIV VSFL VSFR VINT VREL
VFAD VFSB VFML VFDV VAND VOR VNOT VXOR

(2)プレディケイト命令
VGE VGT VLE VLT VEQ VNEQ VFGE VFGT VFLE VFLT VFEQ VFNE

(3)共有メモリアクセス命令
GVLD GVSV GSLD GSSV GELD GESV

(4)レジスタセット命令
IDST FDST SFR SVL

(5)順回演算処理命令
RECR FREC

(6)その他
VWT VFIN

3.3. 実例による各モードでの動作説明

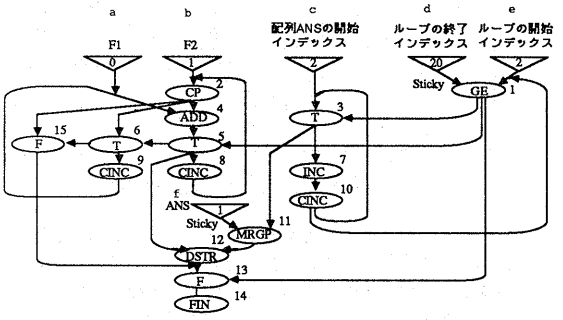
ここでは、ＰＥの各モードにおける動作を実例を挙げ説明する。

3.3.1. データフローモード

次に示すフィボナッチ数列計算プログラム(EX1)を例にとりデータフローモードでのＰＥ内部の動作を説明する。

```
EX1 : /* Fibonacci Sequence */
        F1 = 0
        F2 = 1
        DO 100  N = 2,20
                    FCU = F1 + F2
                    F1 = F2
                    F2 = FCU
                    ANS(N) = F2
100     CONTINUE
```

プログラムＥＸ１をデータフローグラフで表したものを図７に、ＤＭ内のデータ、ＰＭ内のデータをそれぞれ表３，表４に示す。なお簡単のためＥＸ１を１つのマクロブロックとして実現している。



GE(Grater Equql)	左入力>=右入力ならTRUEを，そうでなければFALSEを出力
CP(CoPy)	パケットをコピーする
ADD(ADD)	右入力と左入力との和をとる
T(True-gate)	右入力がTRUEなら左入力を出力
F(False-gate)	右入力がFALSEなら左入力を出力
INC(INCrement)	入力に１を加算する
CINC(Color INCrement)	カラーを１増やす
MRGP(MeRGe Packets)	DSTR命令の右入力を作成する．配列名(左入力)とインデックス(右入力)を結合する
DSTR(Data STo Re to GM)	データ名が右入力であるデータ(左入力)をＧＭに格納する
FIN(FINish)	マクロブロックを終了する

図7 データフローグラフ(EX1)

表3 DM内初期データ(EX1)

	PI	PF	MB	NN	FI	SF	C/D	Op.	DI	Data
a	4	FALSE	1	4	0	FALSE	0	ADD	LEFT	0
b	4	FALSE	1	2	0	FALSE	0	CP	SINGLE	1
c	4	FALSE	1	3	0	FALSE	0	T	LEFT	2
d	4	FALSE	1	1	0	TRUE	0	GE	LEFT	20
e	4	FALSE	1	1	0	FALSE	0	GE	RIGHT	2
f	5	FALSE	1	11	0	TRUE	0	MRGP	LEFT	1

表4 PM内プログラム(EX1)

アドレス	Data flow / Vector	ノード 命令	新NN オペランド1	新MB オペランド2	新PI オペランド3	新DI オペランド4	新命令 オペランド5
1001		1	3	1	4	RIGHT	T
1002		1	5	1	4	RIGHT	T
1003		1	6	1	4	RIGHT	T
1004		1	13	1	4	RIGHT	F
1005		1	15	1	4	LEFT	F
1006		2	4	1	4	RIGHT	ADD
1007		2	6	1	4	LEFT	T
1008		2	15	1	4	LEFT	F
1009		3	7	1	4	SINGLE	INC
1010		3	11	1	5	LEFT	MEGP
1011		4	5	1	4	RIGHT	T
1012		5	8	1	4	SINGLE	CINC
1013		5	12	1	5	RIGHT	DSTR
1014		6	9	1	4	SINGLE	CINC
1015		7	10	1	4	SINGLE	CINC
1016		8	2	1	4	SINGLE	CP
1017		9	4	1	4	RIGHT	ADD
1018		10	1	1	4	LEFT	GE
1019		10	3	1	4	RIGHT	T
1020		11	12	1	5	LEFT	DSTR
1021		12	13	1	4	RIGHT	F
1022		13	14	1	6	SINGLE	FIN
1023		15	13	1	4	LEFT	F

図７において逆三角形は初期データを表し、円はノードを表す。またその横の番号は、ノード番号である。

マクロブロック起動命令がＢＣＮによって受信されると、ＤＭ内のデータ（表３）がＤＭＣによりＷＭに転送される。ＷＭでは、送られて来たパケットに対して相手パケットがＷＭ内に存在するかどうかを調べる。存在すれば対になったパケットをＬＣＵに送出し、存在しなければＷＭに格納する。この場合、表３におけるｄとｅのパケットは同一のマクロブロック番号、ノード番号、カラーを持っており、既に対をなしているのでＷＭからＬＣＵへ送られる。

ＬＣＵへ送られたｄとｅのパケットは、命令が"ＧＥ"であり、ＥＸで実行される命令であるため、そのままＥＸへ送られる。ＥＸでは、ＬＣＵから送られてきたパケットの演算を行う。この場合、命令は"ＧＥ"、左入力が２０、右入力が２であるため、結果としてＴＲＵＥが出力される。次に新しいパケットを作成するために、次のノードに関する情報をＰＭからフェッチする。この場合、ノード番号１の次のノードに対するデータ、すなわちアドレスが1001～1005（表４）のデータがＰＭからＥＸに読み込まれる。例えば、アドレス1001に示されるデータは、次の行き先ノード番号が３、マクロ番号が１、パケット識別子が４、右入力、命令はＴゲートであることを示している。これらのデータに結果のＴＲＵＥが結合されて５つの新しいパケットとなり、ＳＷへ送出される。ノード１では、結果パケットのコピーをＥＸ内で複数のパケットを作ることにより実現している。しかし他のＰＥに対して複数のコピーを送るような場合には、ＰＥ間のデータコミュニケーションによるオーバーヘッドを招く場合がある。したがって、このような場合は１つのパケットを送り出し、受け取り側のＰＥでコピー命令（コピーノード）によりコピーすることが望ましい。

ＳＷでは、ＥＸからのパケットが自ＰＥ宛のものであるかどうかを調べ、自ＰＥ宛のものであればＤＭＣに、そうでなければＤＣＮへ送出する。ＤＭＣは、ＳＷからのパケットが現在活性中のマクロブロックに属するかを調べ、活性パケットであればＷＭへ送出し、そうでなければＤＭへ格納する。

この例の場合は、全て活性パケットであるからWMへ送られる。

このようにして、実行が進み図７に示すノード１４のＦＩＮ命令が実行されると、マクロブロックの実行は終了しＳＷによってその終了がＳＹＭへ報告される。

またＧＭに対するアクセスは、ＤＳＴＲ(Data STaRe to GM)とＤＬＤ(Data LoaD from GM)命令によりＬＣＵで行われる。例えばノード１２は、ANS(N)のデータをＧＭに格納するための命令である。

３．３．２．ベクトルモード

次に示すＥＸ２のプログラムを用いてベクトルモードでのＰＥ内部の動作を説明する。

```
EX2 : /* Sample Program in the Vector mode */
        DO 100  I = 1,N
                A(I) = B(I) + C(I)
100     CONTINUE
```

ＥＸ２に対するＤＭ内データを表５に、ＰＭ内プログラムを表６に、そしてデータフローグラフを図８に示す。

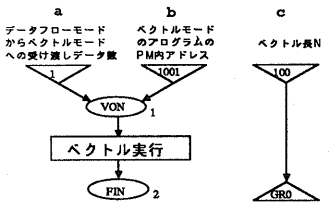


図8 データフローグラフ(EX2)

表5 DM内初期データ(EX2)

	PI	PF	MB	NN	FI	SF	C/D	Op.	DI	Data
a	5	FALSE	1	1	0	FALSE	0	VON	LEFT	1
b	5	FALSE	1	1	0	FALSE	0	VON	RIGHT	1001
c	5	FALSE	1	GR0	0	FALSE	0	RSTR	SINGLE	100

表6 PM内プログラム(EX2)

アドレス	Data flow / Vector	ノード命令	新NN オペランド1	新MB オペランド2	新PI オペランド3	新DI オペランド4	新命令 オペランド5
1000		1	2	1	6	SINGLE	FIN
1001		GVLD	1 (配列B)	1(開始インデックス)	1 (ストライド)	GR0 (データ数)	VR0
1002		GVLD	2 (配列C)	1(開始インデックス)	1 (ストライド)	GR0 (データ数)	VR1
1003		VADD	VR0	VR1	VR2		
1004		VWT					
1005		GVSV	VR2	3 (配列A)	1(開始インデックス)	1 (ストライド)	GR0 (データ数)
1006		VFIN					

ＥＸ２は、ＢＣＮがマクロブロック起動命令を受信することにより、まずデータフローモードとして開始される。

ベクトルモードへの移行は、図８のノード１に示されるＶＯＮ命令をデータフローモードにおいて実行することにより行われる。データフローモードで使用されていたデータをベクトルモードへ受け渡すためには、ＬＣＵ内に用意されている General Resister(GR)を用いる。このＧＲは、図８では三角形で示されている。本例においては、ベクトル長Ｎをデータフローモードからベクトルモードへ受け渡している。一般にデータフローモードからベクトルモードへのデータの受け渡しは複数となるが、この場合はこれらのデータが全てそろった後にベクトルモードの実行を開始しなければならない。このため、ＶＯＮ命令の左入力を受け渡しデータ数とし、この個数分のデータがＧＲに届いていることを確認後、ベクトルモードの実行を開始する。またＶＯＮの右入力は、対応するプログラムの書かれているＰＭ内アドレスである。

ベクトル実行が始まると、まず表６に示すアドレス1001～1002のＧＶＬＤ(Gm Vector LoaD)命令により配列ＢとＣがＬＣＵ内のベクトルレジスタＶＲ０とＶＲ１にそれぞれロードされる。ＧＶＬＤ命令のオペランドは、順番に配列名、index開始値、ストライド 、データ個数、ロード レジスタ名である。

次にＶＡＤＤ命令によりＶＲ０，ＶＲ１間でＡＤＤの計算が行われ、結果がＶＲ２に入る。この時データは、図９で示すようにＥＸから直接ＬＣＵに送られる。

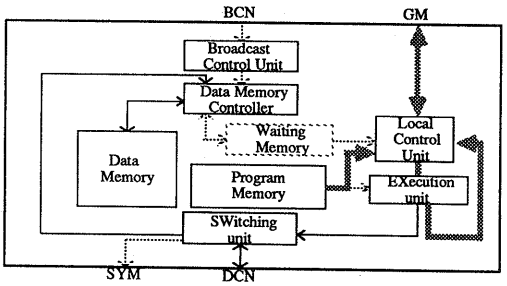


図9 ベクトルモード時のデータの動き

ＧＭへのデータアクセス命令と演算関係の命令は、別々のユニットで実行し、同時実行を可能としている。このため、ベクトル演算の実行と並行して他のデータをＧＭからロードすることが出来る。しかし、この例では演算結果をＧＭへ格納するので、演算と結果格納処理の同時実行は不可能である。このため、次の命令であるＶＷＴ(Vector WaiT)命令を用いＶＡＤＤの終了を待つ。その後ＧＶＳＶ(GM Vector SaVe)命令により結果をＧＭへ格納する。ＧＶＳＶ命令のオペランドは、順番にソースレジスタ名、配列名、index開始値、ストライド、個数、を示す。

最後のＶＦＩＮ(Vector FINish)命令は、ベクトルモードの実行終了命令である。この命令により再びデータフローモードとなり、図８のノード２へ実行が移る。ノード２はＦＩＮ命令であるから、これによって実行が終了する。

３．３．３．両モードの比較

処理対象となるデータがＧＭにある場合、データフローモードではデータ１個１個についてＧＭにアクセスしなければならない。これに対してベクトルモードでは、ＧＭからのデータ取り出し及び格納を高速に処理することにより、処理全体の高速化をはかっているため、データフローモードの約２倍の処理性能があることをシミュレーションにより確認している[1]。

ここでは、ベクトルモードのもうひとつの利点を示す。ベクトルモードでは、演算結果を直接ＬＣＵにフィードバックさせ処理を行っている（図９）。このため、データフローモードにおいて生じるＥＸ→ＷＭ→ＥＸといったパケットのループ通過に伴うオーバーヘッドをなくすことが出来る。特に直前の演算結果により処理が進んでいく場合は、この効果が顕

なる。これを確認するために直前の演算結果を用いて計が進行していく１次の順回演算プログラムＥＸ３を用いて両モードにおける処理時間の比較を－晴－のソフトウエアシミュレータ(1)により行った。なおベクトルモードでは、処理対象となるデータはＧＭにあり、データフローモードではＤＭにあると仮定した。

```
EX3 : /* Recurrence Program */
        DO 100  I = 1,N
                    A(I) = A(I-1) + B(I)
100     CONTINUE
```

結果を図10に示す。

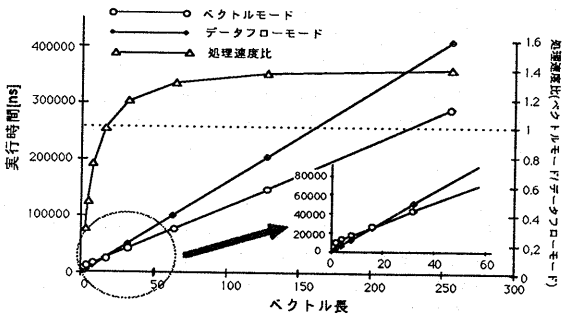


図10 EX3実行時における各モードの処理時間と速度比

ベクトル長Nが約１６を超えるとベクトルモードの方が実行時間が短くなっており、データフローモードに比較すると約１．４倍の処理性能があることがわかる。ＥＸ３のように直前の演算結果を用いて処理が進んで行く場合、データフローモードよりもベクトルモードを用いて処理を行った方が、ＥＸ→WM→ＥＸというループにかかる時間を削減出来るため、高速処理が可能となることがわかった。

以上のようにベクトルモードは、(1)ＧＭ上にあるデータを連続に処理する場合、(2)直前の演算結果を用いて処理が進んで行く場合、に効果がありデータフローモードに比較してそれぞれ約２倍、約１．４倍の処理性能があることを確認した。

４．待ち合わせ記憶WMの構成とその動作

データフロー実行では、待ち合わせ記憶の高速化がシステム全体の高速化において重要なポイントとなる。ここでは、－晴－における待ち合わせ記憶の(1)バンク化、(2)裏バンクへのパケットプリフェッチ、による高速化について示す。

４．１．構成と動作

待ち合わせ記憶の構成に関しては、これまでにハードウエア量を抑え高速にこれを実現する方法としてハッシュ法に改良を加えたもの(4)などが提案されている。しかしながら、ハッシュ法で待ち合わせ記憶（連想記憶）を実現した場合と、フルアソシアティブにこれを構成した場合では、速度の面で後者の方が有利であり、特に容量が数百パケット程度であればハードウエア量を考えてもフルアソシアティブに構成した方が有利であるという報告(5)がされている。－晴－では特に、マクロブロック化することによってWM内のパケット数を少なく抑えることが出来、容量を小さく出来る(2)のでハッシュ法よりもフルアソシアティブな構成の方が有利である。したがって－晴－では、この２つの方式の内フルアソシアティブな構成をとる。しかしながら、フルアソシアティブな構成のみでは、方式的にこれ以上の高速化は望めない。そこで図11に示すように(1)WMのバンク化及び(2)裏バンク化を行い高速処理を実現する。

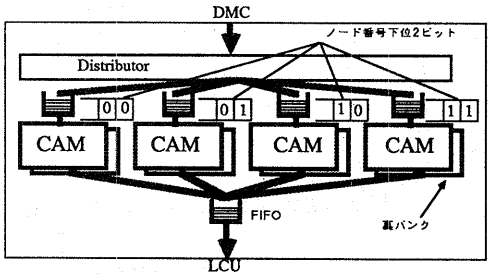


図11 バンク化及び裏バンク化をしたWM

各バンクの使用は、WMに入って来るそれぞれのパケットの実行ノード番号下位２ビットによって決定する。例えば、下位２ビットが"00"であれば 一番左の連想記憶(CAM)を、"01"であればその次のＣＡＭ、といった具合いである。このようにすることで、パケットが平均的に到着した場合、１つのＣＡＭに比較して最大４倍のスループット向上が期待できる。

次にWMの裏バンクについて説明する。－晴－においては不活性パケットはＤＭに格納し、マクロブロック起動の時点でそれらのパケットをWMに転送するという方式をとっている。このため、ＤＭ→WMの転送によるオーバーヘッドがマクロブロック起動毎にかかる。このオーバーヘッドを隠すためにWMに裏バンクを設け、起動前に転送しておく方式をとる。この方式によれば、マクロブロックの起動と同時にバンク切り換えを行うだけで実行を開始出来る。特に、あらかじめそのマクロブロックを起動することがわかっている場合や、条件分岐によりそのマクロブロックの起動の確率が高い場合においては、この裏バンクへのプリフェッチ効果が大きい。

条件分岐などで使用されなくなったいわゆるゴミパケットは、各マクロブロック終了後にWMを初期化することにより行う。このため、ゴミパケット数が１マクロブロック実行中にWMをオーバーフローさせない範囲においては、ゴミパケット回収のためのセルフクリーニングなプログラムを書く必要がなく、処理の高速化にもつながる。また、WMがオーバーフローした場合は、ＤＭに一時的にオーバーフローしたパケットを格納する。

４．２．WMのシミュレーションによる評価

ここでは、WMのバンク化による効果と裏バンクへのプリフェッチ効果について簡単な評価を行う。

４．２．１．バンク方式による効果

WMのバンク化による効果を確認するために、ＧＰＳＳを用いて(1)４バンク時、(2)２バンク時、(3)１バンク時、の場合について、WMにパケットが到着してから処理を終え、出るまでの時間（WM系内通過時間）を求め比較を行った。シミュレーションでは、図11のＣＡＭの処理時間を 400(ns)、distributor及び queueの処理に 50(ns)かかると仮定した。また、(3)１バンク時は、distributor及び出力の queueは必要ないので存在しないものとして考えた。パケットの到着はポアソン分布であるとし、その到着間隔の平均値を変化させ

たときのWM系内通過時間の変化を図12に示す。なお、網かけ部分は、WM系内通過時間の分散（平均±標準偏差）を示している。

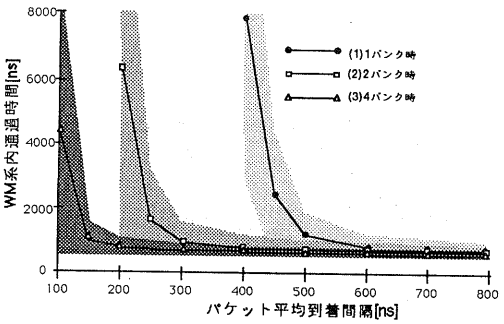


図12 WMのバンク化による処理性能の向上

図12によると、バンク化した場合、パケットの平均到着間隔が短くなってもWM系内通過時間を増加させることなく処理出来ることがわかる。WM系内通過時間平均が 1000〔ns〕の点におけるパケット平均到着間隔を３つの場合について比較すると、１バンク時を基準として、２バンク、４バンク時にそれぞれ約1.7倍、約3.4倍の処理性能があることがわかる。さらに、WM系内通過時間平均が 1000〔ns〕の点におけるWM系内通過時間の分散を比較すると、４バンクの時が一番小さくなっており、処理時間のばらつきを小さく出来ることがわかる。

以上より、バンク方式によってWM処理性能の向上がはかれることを確認した。

4.2.2. 裏バンクによるプリフェッチ効果

裏バンクへのプリフェッチ効果を調べるために－晴－のソフトウエアシミュレータによりシミュレーションを行った。使用したプログラムは２つのマクロブロックからなり、それぞれの演算は全く同じものとし、１台のＰＥで実行するものとした。そして各マクロブロック内のパケット数を変化させた時の(1)プリフェッチをしない場合、(2)プリフェッチをした場合、(3)２つのマクロブロックを１つのマクロブロックとして実行した場合、の３つの場合について実行時間を比較した。(1)の実行時間を１とした時の(2)及び(3)の実行時間比を図13に示す。

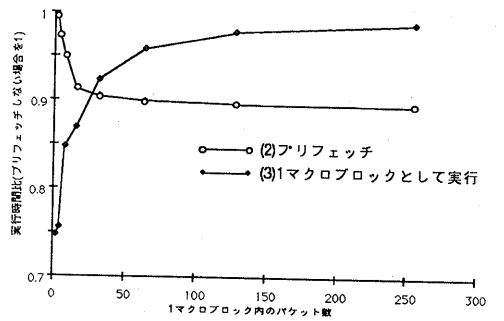


図13 裏バンクへのプリフェッチ効果

パケット数が小さい場合には、(3)の場合が(1)の場合に比較して実行時間が短くなっている。これは(3)の場合では、マクロブロック起動が１回でよいため、起動のオ[illegible]ドが削減されたためである。しかし、パケット数[illegible]に伴いマクロブロック起動にかかる時間の割合が全[illegible]時間に比べて小さくなるために、(1)の場合とほとん[illegible]らなくなる。

これに対して(2)の場合は、パケット数の増加に伴っ[illegible]理時間が(1)の場合に比べて減少している。これは、第[illegible]クロブロックの実行と並行して裏バンクに第二マクロブロックのパケットを転送するために、ＤＭ→WMのパケット転送のオーバーヘッドを隠すことが出来るからである。また、同じ理由でパケット数が約３０のところで(3)の場合に比較しても処理時間が短くなっている。ただし３０までは、マクロブロック起動のオーバーヘッドにより、(3)の場合の方が(2)の場合より早くなっている 。

以上、裏バンクにプリフェッチを行うことにより、プリフェッチしない場合に比較して約１割の実行時間短縮が出来ることがわかった。この実行時間短縮率はプログラムに大きく依存するため、今後いろいろな場合について検討していく。

5．おわりに

並列処理システム－晴－の要素プロセッサ構成に関して、その実行制御機構を中心に述べた。要素プロセッサ内の処理の高速化の手法として、(1)ベクトルモード処理、(2)待ち合わせ記憶のフルアソシアティブな構成とバンク化、(3)待ち合わせ記憶の裏バンクへのプリフェッチ、について述べると共に、評価を行いその有効性を示した。

今後は、さらに詳細な設計を行い要素プロセッサのプロトタイプの製作にとりかかる予定である。

謝辞

シミュレーションプログラム作成及び本研究の遂行にあたり御討論いただいた本研究室の萩原 孝氏に感謝いたします。

参考文献

〔1〕丸島,山名,萩原,草野,村岡："並列処理システム－晴－の実行方式",情処研報,88-CA-69,pp.9-16 (1988)

〔2〕山名,丸島,萩原,村岡:"科学技術計算用並列処理システム－晴－の要素プロセッサ構成の検討",情処第34回全大,1Q-8 (1987)

〔3〕草野,山名,丸島,村岡:"並列処理システム－晴－における要素プロセッサのシミュレーション評価",情処第36回全大,1C-1 (1988)

〔4〕平木,西田,島田:"連鎖並列ハッシングを用いた連想記憶方式の評価",信学技報,EC-82-71,pp.31-42 (1982)

〔5〕西田,平木,島田:"データ駆動計算機用マッチングユニットのＬＳＩ化の検討",信学技報,EC-83-24,pp.31-38 (1983)